Camera Calibration Using a Box

Computer Vision Project

Yongjik Kim & Joohi Lee

Spring 2000

Introduction

The goal of this project is to calibrate camera with a box which is painted white to reduce error.
We used two cameras fixed on the HMD for calibration.

First, we took stereo image of a box and computed intrinsic/extrinsic matrices from them.
Transform between two camera is also obtained.
Then, with another stero image from different view point, we did:
1. detect vertices from the left image
2. transform them according to the intrinsic matrices & left-to-right transform matrix
3. compare the transformed vertex coordinates with obtained right image

---

Steps

Edge detection

We detected edge by simply calculating its gradiant at each pixel. Edge magnitude is calculated as `sqrt(I_x^2 + I_y^2)`.

Edge angle map is also calculated, but here we first filter the image with Gaussian filter with `sigma=2`. (Otherwise, the angle map contains too much noise to be useful.) Then the angle is calculated as `atan2(I_y, I_x)`.

Hough transform

Since we have edge information, we don't need to draw a sine curve for each pixel. Instead, for each pixel, we find a corresponding point in the Hough transform map, and increase the intensity of the point by edge magnitude.

The Hough transform map is indexed by `phi` and `rho`. Each point of the HT map corresponds to a line `x*cos(phi)+y*sin(phi) = rho`, where the origin `(x,y) = (0,0)` is the center of the image.

The range of `phi` is between 0 and `2pi`, with 120 steps (each step is 3^o wide). The range of `rho` is between 0 and (maximum possible value = half of the diagonal of image), where each step is 2 pixels wide.

After Hough transform is done, we filter the **Hough transformed map** with Gaussian filter of `sigma=1` to reduce noise, and find all the local maxima. (If we don't confine ourselves into local maxima, strong edges would generate so many spurious neighboring edges that it would be very hard to suppress them later.)

Each found local maxima indicates a candidate edge, and its intensity indicates how strong the edge is.

Line fitting

Since Hough transform is done by generating table of finite grid size, its result inevitably deviates from the 'best' result we can find from the image. To solve this problem, we do the following for each found edges, from the strongest one to the weakest one.

- We first select all the 'candidate' edge points that is **1)** within 5 pixels from the given edge and has its edge angle within 7.5^o from the given edge angle.
- We then do a linear least square fitting to find a line that best approximates these points.
- We then again find all the points that is within 5 pixels from the newly calculated edge and has its edge angle 7.5^o from the edge angle. We *disable* all these points. They are considered to *belong* to this edge, and is not considered in fitting any subsequent edges.

  * We did it in order to *discriminate* spurious edges. If there are several edges that corresponds one real edge, ideally, the first edge to be considered would *take* all the edge points, and other edges would find no edge points to support them and be discarded. Although it worked in some cases, it wasn't always reliably working. We think we need more sophisticated methods to detect and eliminate spurious edges.
- We sum all the edge magnitude from those points that belong to this edge, and assign this value into the magnitude of newly fitted edge. (If the value is below certain threshold, the edge is discarded altogether.)

Finding intersection

Now we have the set of all the re-fitted edges, with its 'intensity' values. We consider the intersection between *any* pair of these points, again from the strongest to the weakest edges.

This problem is not easy. If we have spurious edges, they will generate spurious intersection points. Even if we don't have spurious edges, three real edges can meet at a vertex of a box, which will generate three intersection points. Moreover, two perfectly correct edges that don't meet in the image can have mathematical intersection point that doesn't exist in the real image.

The simple solution we adopted is to discard all the points that is closer to any already found point than 10 pixels. This is a value selected by trial and error, but it's not always working. (If we make it small, it will generate more and more spurious points; on the other hand, big value will make the location of points more and more inaccurate, and may miss a real intersection point if it is too close to another real intersection.)

For each pair of lines, we must decide whether these two lines indeed meet in the image. To find it, we generated a simple *salience function*. It is calculated as:

$\texttt{salience} = \texttt{min}_{\text{two edges}}$ (total edge strength) * (sum of edge magnitude that belongs to this edge and falls within 20 pixels of the intersection point)

Obviously, this is rather an *ad hoc* criterion, but it seems to work if the image is very good. (If the image is *not very good*, it still works but we also have lots of wrong intersection points that are ranked higher than real intersection points. Therefore, it becomes very hard to figure out which point corresponds to which vertex in the box.)

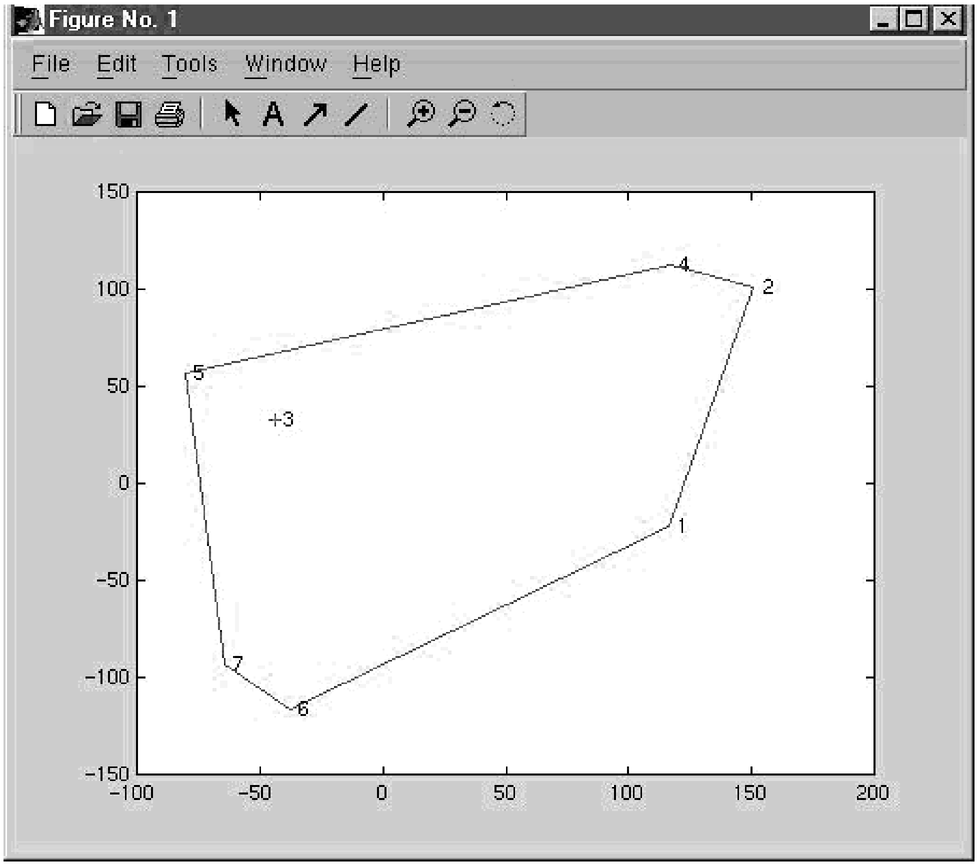
Correspondence

For camera calibration, 6 pairs of world/image points are needed.
Here, the problem arises : How can we select corresponding pairs?

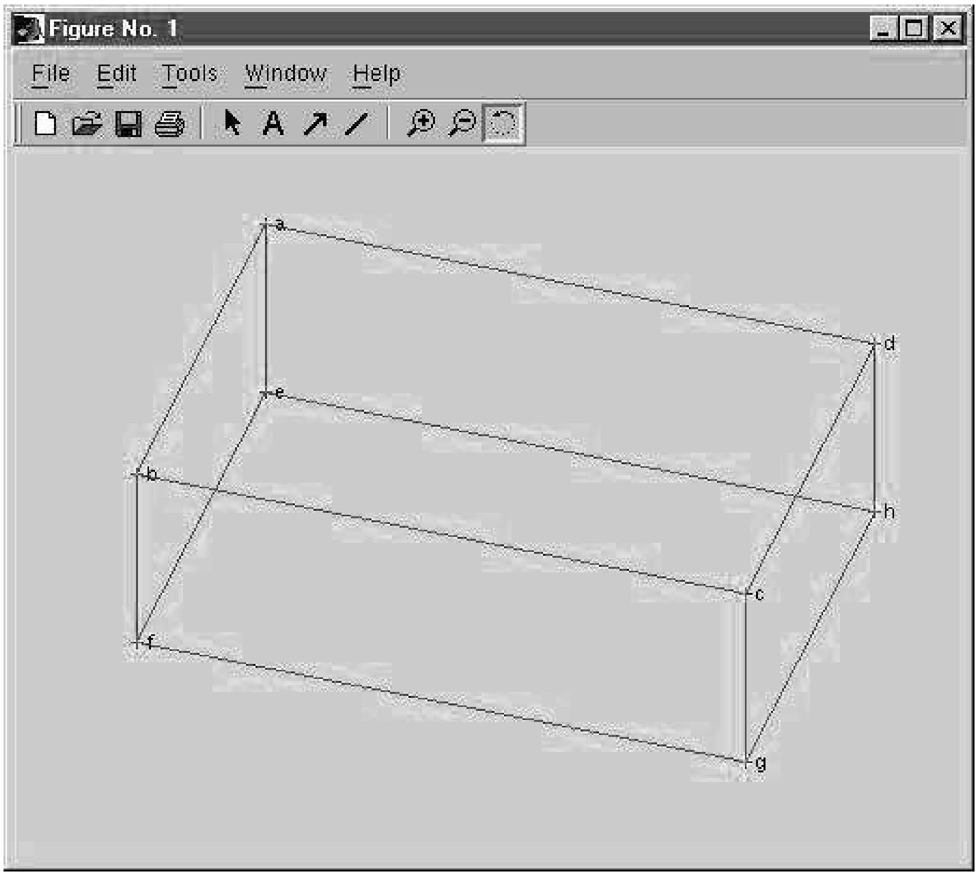
For the first time of calibration, we manually selected corresponding pairs.

But, once intrinsic parameters are known, we can exploit the parameters, symmetry of a box and convex hull property.
Following is a subroutine to find correspondence.

> **Given 7 intersections in image, find convex hull of them.**
>
> **Map a point which is totally inside of the convex hull to an arbitrary vertex of a box.**
>
> **Calibrate intrinsic parameters for 6 possible combination of correspondence.**
>
> **Find the combination whose intrinsic parameters are closest to the one that we obtained before.**



Seven points and the convex hull
Point 6 is located inside of the convex hull



geometry of a box
Map 3 to c (arbitrary point)
Correct correspondence is

a - 1, b - 2, d - 6, f - 4, g - 5, h - 7
6 combinations to be tried are (for a,b,d,f,g,h)
1,2,6,4,5,7
2,6,4,5,7,1
6,4,5,7,1,2
4,5,7,1,2,6
5,7,1,2,6,4
7,1,2,6,4,5

Compute intrinsic parameter for each corresponding pair and find best match.

For 14 strongest intersections, choose 7 out of them, and do above procedure.
Overall the best match gives corresponding pairs.

Camera Calibration

From 6 pairs of image coord. and world coord. points, find 3x4 projection matrix that satisfies,

$$\begin{pmatrix} u_i^{image} \\ v_i^{image} \\ w_i^{image} \end{pmatrix} = M \begin{pmatrix} X_i^{world} \\ Y_i^{world} \\ Z_i^{world} \\ 1 \end{pmatrix}$$

M gives intrinsic and extrinsic parameters of camera.
What we get finally is,

fx, fy	focal lenth devide by scale factor. scale factor is dependent on world coordinate system, size of pixel in CCD camera
Ox, Oy	coordinate of image center(principal point) in pixel
R	rotation of camera from world coord. to image coord.
T	translation of camera from world coord. to image coord.
Mint Mext	intrinsic/extrisic matrix $M_{int} = \begin{pmatrix} -f_x & 0 & O_x \\ 0 & -f_y & O_y \\ 0 & 0 & 1 \end{pmatrix} M_{ext} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{pmatrix}$

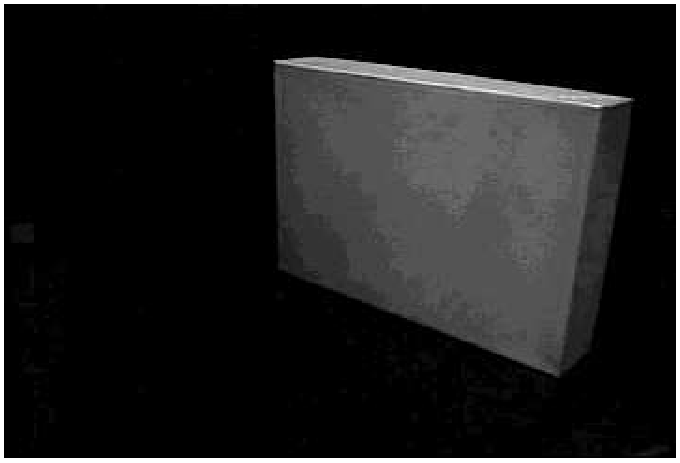
Two extrinsic matrice of each camera also give Left to Right transform.

More detail about calibration can be found at <Introduction to Computer Vision> Chapter 6.3 by Trucco & Verri

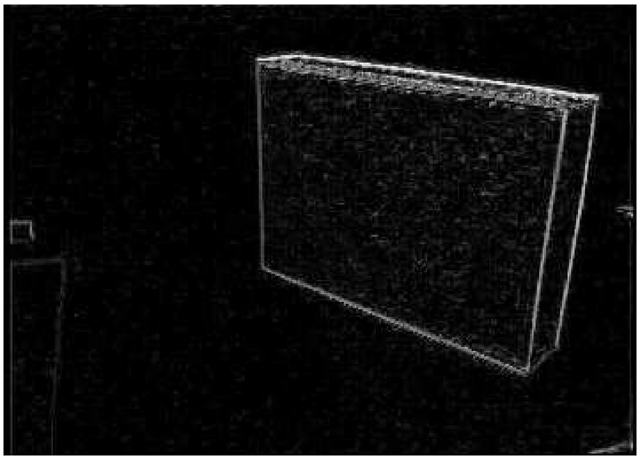
---

Results

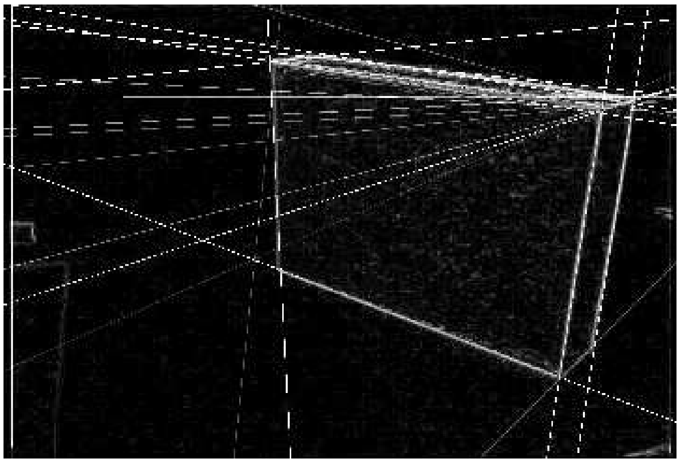
* click the images to see full-sized version



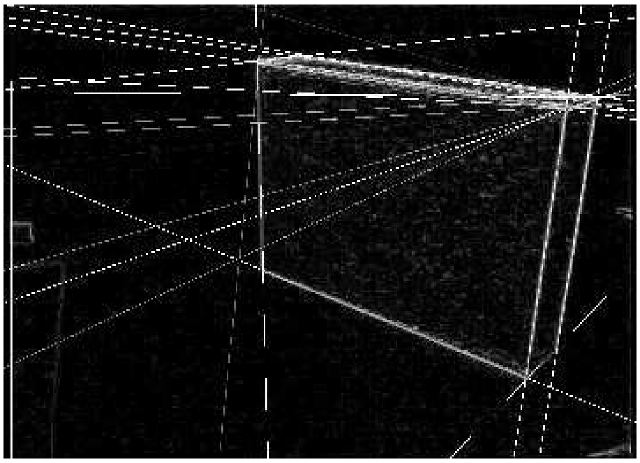
original image taken by left camera



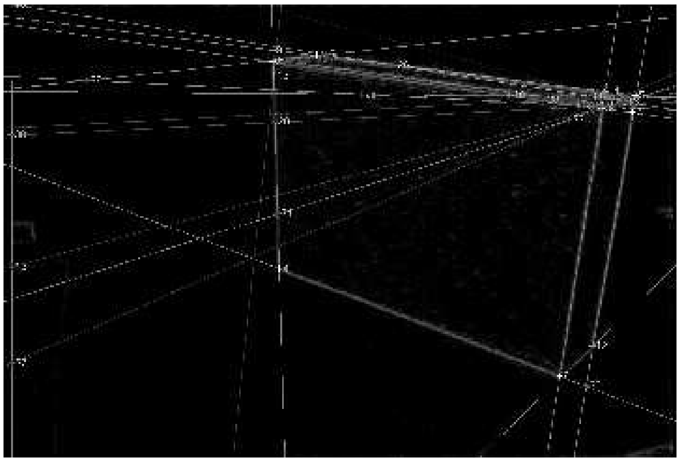
edge detection



edges found from Hough transform



Line fitting



intersection points